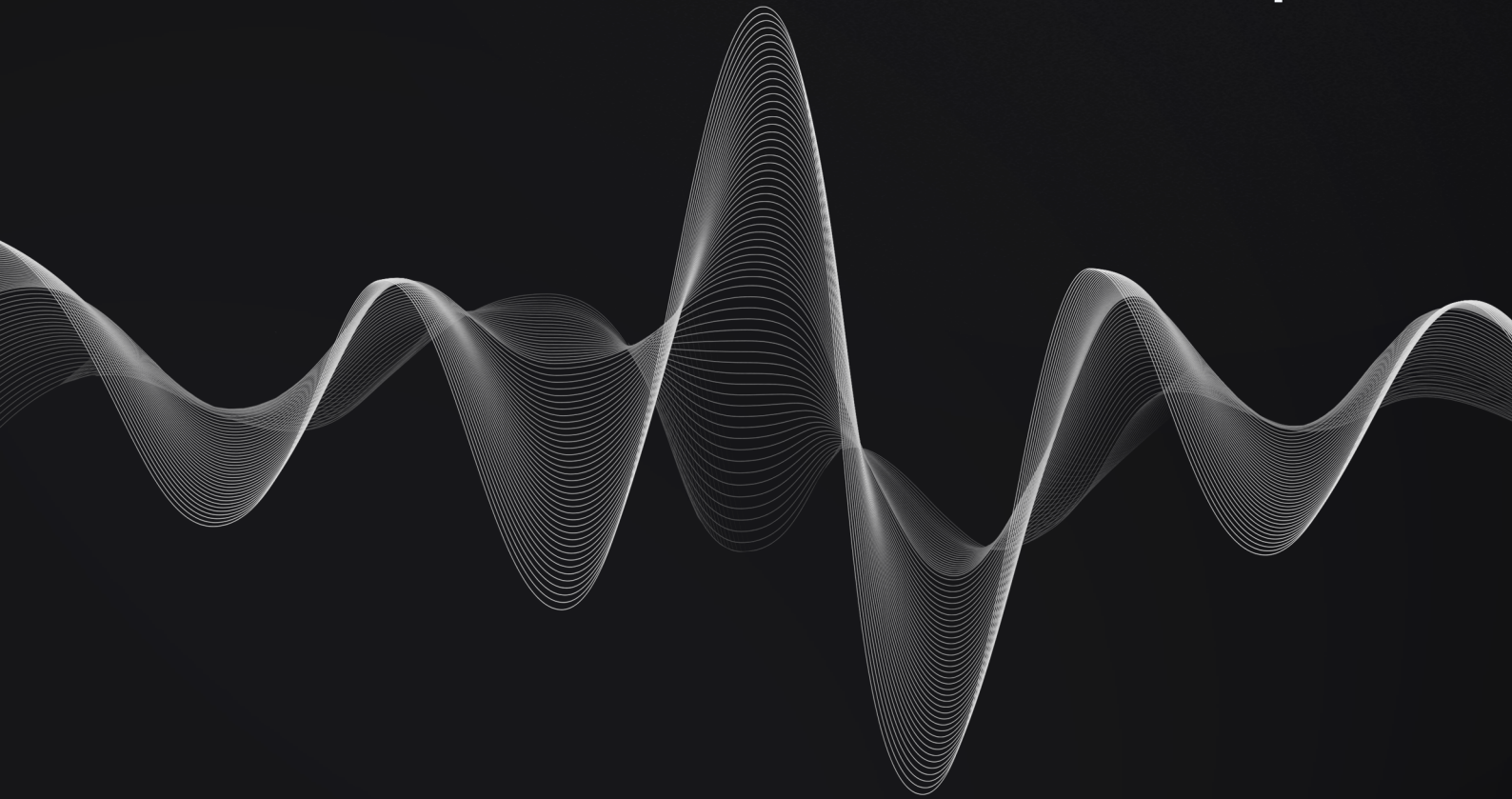


**LivLive<sup>+</sup>**

**LivLive**

# LivLive Smart Contract Audit Report











# Document Control

**PUBLIC**

**FINAL**(v2.2)

## Audit\_Report\_LIV-NFT\_FINAL\_22

Jun 10, 2025		<b>v0.1</b>	Luis Arroyo: Initial draft
Jun 11, 2025		<b>v0.2</b>	Luis Arroyo: Added findings
Jun 11, 2025		<b>v0.3</b>	João Simões: Added findings
Jun 11, 2025		<b>v1.0</b>	Charles Dray: Approved
Jun 12, 2025		<b>v1.1</b>	João Simões: Reviewed findings
Jun 24, 2025		<b>v2.0</b>	Charles Dray: Finalized
Jul 15, 2025		<b>v2.1</b>	Charles Dray: Published
Sep 18, 2025		<b>v2.2</b>	João Simões: Updated point of contact

<b>Points of Contact</b>	Elvin Adrian Charles Dray	LivLive Resonance	contact@livlive.com charles@resonance.security
<b>Testing Team</b>	Luis Arroyo João Simões Michał Bazyli Ilan Abitbol	Resonance Resonance Resonance Resonance	luis.arroyo@resonance.security joao@resonance.security michal@resonance.security ilan@resonance.security



## Copyright and Disclaimer

© 2025 Resonance Security, Inc. All rights reserved.

The information in this report is considered confidential and proprietary by Resonance and is licensed to the recipient solely under the terms of the project statement of work. Reproduction or distribution, in whole or in part, is strictly prohibited without the express written permission of Resonance.

All activities performed by Resonance in connection with this project were carried out in accordance with the project statement of work and agreed-upon project plan. It's important to note that security assessments are time-limited and may depend on information provided by the client, its affiliates, or partners. As such, the findings documented in this report should not be considered a comprehensive list of all security issues, flaws, or defects in the target system or codebase.

Furthermore, it is hereby assumed that all of the risks in electing not to remedy the security issues identified henceforth are sole responsibility of the respective client. The acknowledgement and understanding of the risks which may arise due to failure to remedy the described security issues, waives and releases any claims against Resonance, now known or hereafter known, on account of damage or financial loss.

# Contents

<b>1 Document Control</b>	<b>2</b>
Copyright and Disclaimer .....	2
<b>2 Executive Summary</b>	<b>4</b>
System Overview .....	4
Repository Coverage and Quality.....	4
<b>3 Target</b>	<b>5</b>
<b>4 Methodology</b>	<b>6</b>
Severity Rating.....	7
Repository Coverage and Quality Rating.....	8
<b>5 Findings</b>	<b>9</b>
Solidity Version 0.8.20 Incompatibilities Due To PUSH0.....	10
Update Import Usages To Add Modularity.....	11
Floating Pragma .....	12
Array Increment Should Be Unchecked .....	13
uint Should Not Be Initialized With 0 .....	14
Fallback Mechanism In Case Of Failed NFT Mint .....	15
Redundant Code.....	16
<b>A Proof of Concepts</b>	<b>17</b>

# Executive Summary

**LivLive** contracted the services of Resonance to conduct a comprehensive security audit of their smart contracts between June 09, 2025 and June 11, 2025. The primary objective of the assessment was to identify any potential security vulnerabilities and ensure the correct functioning of smart contract operations.

During the engagement, Resonance allocated 2 engineers to perform the security review. The engineers, including an accomplished professional with extensive proficiency in blockchain and smart-contract security, encompassing specialized skills in advanced penetration testing, and in-depth knowledge of multiple blockchain protocols, devoted 2 days to the project. The project's test targets, overview, and coverage details are available throughout the next sections of the report.

The ultimate goal of the audit was to provide LivLive with a detailed summary of the findings, including any identified vulnerabilities, and recommendations to mitigate any discovered risks. The results of the audit are presented in detail further below.

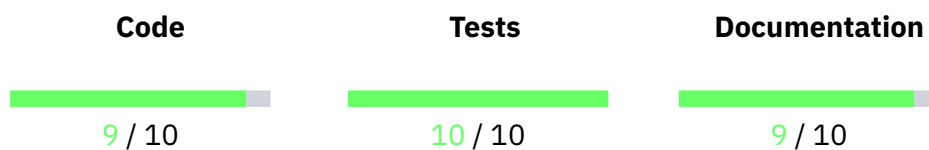


## System Overview

The LivLive system is a dual-payment NFT minting platform that supports both fiat (via Wert.io) and crypto (via NowPayments) transactions. It also implements a revenue distribution smart contract that manages a 2-level referral commission system.



## Repository Coverage and Quality



Resonance's testing team has assessed the Code, Tests, and Documentation coverage and quality of the system and achieved the following results:

- The code follows development best practices and makes use of known patterns, standard libraries, and language guides. It is easily readable and uses the latest stable version of relevant components. Overall, **code quality is excellent**.
- Unit and integration tests are included. The tests cover both technical and functional requirements. Code coverage is 100%. Overall, **tests coverage and quality is excellent**.
- The documentation includes the specification of the system, technical details for the code, relevant explanations of workflows and interactions. Overall, **documentation coverage and quality is excellent**.

# Target

The objective of this project is to conduct a comprehensive review and security analysis of the smart contracts that are contained within the specified repository.

The following items are included as targets of the security assessment:

- Repository: [LivLive-Web3/shop](#)
- Hash: 92f7b3c7a0b76fd6a7fd2fef6da2fe1af6bd99e

The following items are excluded:

- External and standard libraries
- Files pertaining to the deployment process
- Financial related attacks

# Methodology

In the context of security audits, Resonance's primary objective is to portray the workflow of a real-world cyber attack against an entity or organization, and document in a report the findings, vulnerabilities, and techniques used by malicious actors. While several approaches can be taken into consideration during the assessment, Resonance's core value comes from the ability to correlate automated and manual analysis of system components and reach a comprehensive understanding and awareness with the customer on security-related issues.

Resonance implements several and extensive verifications based off industry's standards, such as, identification and exploitation of security vulnerabilities both public and proprietary, static and dynamic testing of relevant workflows, adherence and knowledge of security best practices, assurance of system specifications and requirements, and more. Resonance's approach is therefore consistent, credible and essential, for customers to maintain a low degree of risk exposure.

Ultimately, product owners are able to analyze the audit from the perspective of a malicious actor and distinguish where, how, and why security gaps exist in their assets, and mitigate them in a timely fashion.

## Source Code Review - Solidity EVM

During source code reviews for Web3 assets, Resonance includes a specific methodology that better attempts to effectively test the system in check:

1. Review specifications, documentation, and functionalities
2. Assert functionalities work as intended and specified
3. Deploy system in test environment and execute deployment processes and tests
4. Perform automated code review with public and proprietary tools
5. Perform manual code review with several experienced engineers
6. Attempt to discover and exploit security-related findings
7. Examine code quality and adherence to development and security best practices
8. Specify concise recommendations and action items
9. Revise mitigating efforts and validate the security of the system

Additionally and specifically for Solidity EVM audits, the following attack scenarios and tests are recreated by Resonance to guarantee the most thorough coverage of the codebase:

- Reentrancy attacks
- Frontrunning attacks
- Unsafe external calls
- Unsafe third party integrations
- Denial of service
- Access control issues

- Inaccurate business logic implementations
- Incorrect gas usage
- Arithmetic issues
- Unsafe callbacks
- Timestamp dependence
- Mishandled panics, errors and exceptions

## Severity Rating

Security findings identified by Resonance are rated based on a Severity Rating which is, in turn, calculated off the **impact** and **likelihood** of a related security incident taking place. This rating provides a way to capture the principal characteristics of a finding in these two categories and produce a score reflecting its severity. The score can then be translated into a qualitative representation to help customers properly assess and prioritize their vulnerability management processes.

The **impact** of a finding can be categorized in the following levels:

1. Weak - Inconsequential or minimal damage or loss
2. Medium - Temporary or partial damage or loss
3. Strong - Significant or unrecoverable damage or loss

The **likelihood** of a finding can be categorized in the following levels:

1. Unlikely - Requires substantial knowledge or effort or uncontrollable conditions
2. Likely - Requires technical knowledge or no special conditions
3. Very Likely - Requires trivial knowledge or effort or no conditions

		Likelihood		
		Very Likely	Likely	Unlikely
Impact	Strong	Critical	High	Medium
	Medium	High	Medium	Low
	Weak	Medium	Low	Info





# Repository Coverage and Quality Rating

The assessment of Code, Tests, and Documentation coverage and quality is one of many goals of Resonance to maintain a high-level of accountability and excellence in building the Web3 industry. In Resonance it is believed to be paramount that builders start off with a good supporting base, not only development-wise, but also with the different security aspects in mind. A product, well thought out and built right from the start, is inherently a more secure product, and has the potential to be a game-changer for Web3's new generation of blockchains, smart contracts, and dApps.

Accordingly, Resonance implements the evaluation of the code, the tests, and the documentation on a score **from 1 to 10** (1 being the lowest and 10 being the highest) to assess their quality and coverage. In more detail:

- Code should follow development best practices, including usage of known patterns, standard libraries, and language guides. It should be easily readable throughout its structure, completed with relevant comments, and make use of the latest stable version components, which most of the times are naturally more secure.
- Tests should always be included to assess both technical and functional requirements of the system. Unit testing alone does not provide sufficient knowledge about the correct functioning of the code. Integration tests are often where most security issues are found, and should always be included. Furthermore, the tests should cover the entirety of the codebase, making sure no line of code is left unchecked.
- Documentation should provide sufficient knowledge for the users of the system. It is useful for developers and power-users to understand the technical and specification details behind each section of the code, as well as, regular users who need to discern the different functional workflows to interact with the system.



# Findings

During the security audit, several findings were identified to possess a certain degree of security-related weaknesses. These findings, represented by unique IDs, are detailed in this section with relevant information including Severity, Category, Status, Code Section, Description, and Recommendation. Further extensive information may be included in corresponding appendices should it be required.

An overview of all the identified findings is outlined in the table below, where they are sorted by Severity and include a **Remediation Priority** metric asserted by Resonance's Testing Team. This metric characterizes findings as follows:

- ||||| **"Quick Win"** Requires little work for a high impact on risk reduction.
- |||| **"Standard Fix"** Requires an average amount of work to fully reduce the risk.
- ||| **"Heavy Project"** Requires extensive work for a low impact on risk reduction.

---

RES-01	Solidity Version 0.8.20 Incompatibilities Due To PUSH0		Acknowledged
RES-02	Update Import Usages To Add Modularity		Resolved
RES-03	Floating Pragma		Resolved
RES-04	Array Increment Should Be Unchecked		Resolved
RES-05	uint Should Not Be Initialized With 0		Resolved
RES-06	Fallback Mechanism In Case Of Failed NFT Mint		Acknowledged
RES-07	Redundant Code		Resolved



# Solidity Version 0.8.20 Incompatibilities Due To PUSH0

Info

RES-LIV-NFT01

Data Validation

Acknowledged

## Code Section

- [LivLiveNFT.sol#L2](#)
- [LivLiveRevenue.sol#L2](#)

## Description

The compiler for Solidity 0.8.20 switches the default target EVM version to Shanghai, which includes the new PUSH0 op code. This op code may not yet be implemented on all L2s, so deployment on these chains may fail.

The command `forge build -force -extra-output evm.bytecode.opcodes` will print the used opcodes in `out/LivLiveNFT.sol/LivLiveNFT.json`, where it is possible to search for the push0 opcode.

## Recommendation

It is recommended to use an earlier as described in [SolidityLang](#), should the need arise, to deploy to L2 blockchains that do not support this opcode.

## Status

*The issue was acknowledged by LivLive's team. The development team stated "Contracts will be deployed on Ethereum mainnet (L1)".*



# Update Import Usages To Add Modularity

Info

RES-LIV-NFT02

Code Quality

Resolved

## Code Section

- [LivLiveNFT.sol#L4-L11](#)
- [LivLiveRevenue.sol#L4-L8](#)

## Description

Two different syntaxes for importing contracts from external libraries or dependencies are used in Solidity:

1. `import "@openzeppelin/contracts/token/ERC20/ERC20.sol";`
2. `import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol"`

Second syntax allows developers to selectively import specific contracts from a file and provides finer control over which contracts are being imported. This could be useful for reducing namespace clutter or importing only necessary code for the contract.

## Recommendation

It is recommended to update imports with curly braces usage if possible.

## Status

*The issue has been fixed in [ba15cf2dbf4d32d5c9687f574caa743e200a2aed](#).*



# Floating Pragma

Info

RES-LIV-NFT03

Code Quality

Resolved

## Code Section

- [LivLiveNFT.sol#L2](#)
- [LivLiveRevenue.sol#L2](#)

## Description

The project uses floating pragmas `^0.8.20`.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

## Recommendation

It is recommended to use a strict and locked pragma version for solidity code. Preferably, the version should be neither too new or too old.

## Status

*The issue has been fixed in [c3b2ec8087714fccf42f9080e47c6c27d275ab04](#).*



# Array Increment Should Be Unchecked

Info

RES-LIV-NFT04

Gas Optimization

Resolved

## Code Section

- [LivLiveNFT.sol#L216](#)
- [LivLiveRevenue.sol#L123](#)

## Description

The `unchecked` keyword is new in solidity version 0.8.0, so this only applies to that version or higher, which these instances are. This saves 30-40 gas per loop.

## Recommendation

It is recommended to increment the counter `i` in an unchecked snippet:

```
unchecked { ++i; }
```

## Status

*The issue has been fixed in [7d7dab4b60b8da2f4be4f249a5e43aaa232f764e](#).*



# uint Should Not Be Initialized With 0

Info

RES-LIV-NFT05

Gas Optimization

Resolved

## Code Section

- [LivLiveNFT.sol#L216](#)
- [LivLiveRevenue.sol#L123](#)

## Description

Initializing variables to their default values spends gas unnecessarily.

## Recommendation

It is recommended to not initialize variables to their default values, in this case 0.

## Status

*The issue has been fixed in [f06bb4f231805e3fadebb337957be5972026a301](#).*



# Fallback Mechanism In Case Of Failed NFT Mint

Info

RES-LIV-NFT06

Business Logic

Acknowledged

## Code Section

- [LivLiveNFT.sol#L184](#)
- [LivLiveNFT.sol#L218](#)

## Description

During minting of the NFT through the calls to `_safeMint()`, it may be possible that the receiver of the NFT is a smart contract that did not implement the function `onERC721Received()`. In these cases, the minting of the NFT will fail, and depending on the payment option used by the user (Wert.io or NowPayments), fallback or refund mechanisms should be put into place. The users may be left without funds and without the NFT.

## Recommendation

It is recommended to implement automatic or manual fallback or refund mechanisms for the case where minting of the NFT to a smart contract fails. These mechanisms may be implemented on or off-chain.

## Status

*The issue was acknowledged by LivLive's team. The development team stated "We are aware of this. The backend or Wert are minting the NFTs. If minting through Wert fails (Fiat) the Funds are automatically sent to the user (basically refunded) and payment is aborted. If minting through NowPayments fails (Crypto) the payment goes through regularly as users don't really buy because of the NFT but for the physical wristband. This is being monitored and will be handled as support case as we can mint infinite NFTs."*





# Redundant Code

Info

RES-LIV-NFT07

Code Quality

Resolved

## Code Section

- [LivLiveNFT.sol#L95-L96](#)

## Description

The function calls `_setRoleAdmin()` within the `constructor()` are redundant since the role `DEFAULT_ADMIN_ROLE` is already the admin of every role by default.

## Recommendation

It is recommended to remove redundant code to improve code readability and composability.

## Status

*The issue has been fixed in [e8c766183211f8da3a2f40114c457d8c2f1bee44](#).*

# Proof of Concepts

*No Proof-of-Concept was deemed relevant to describe findings in this engagement.*